SUPLEMENTO MENSUAL DEDICADO A LA IMAGEN SINTÉTICA

RenderManía

Número 3

TALLER VIRTUAL

Nuevas versiones de Moray y Breeze

BIBLIOTECA 3D

Modelado por intersecciones

EL FORO

Chicas manga, animaciones, aviones... lo mejor de nuestros artistas

EN EL CD-ROM

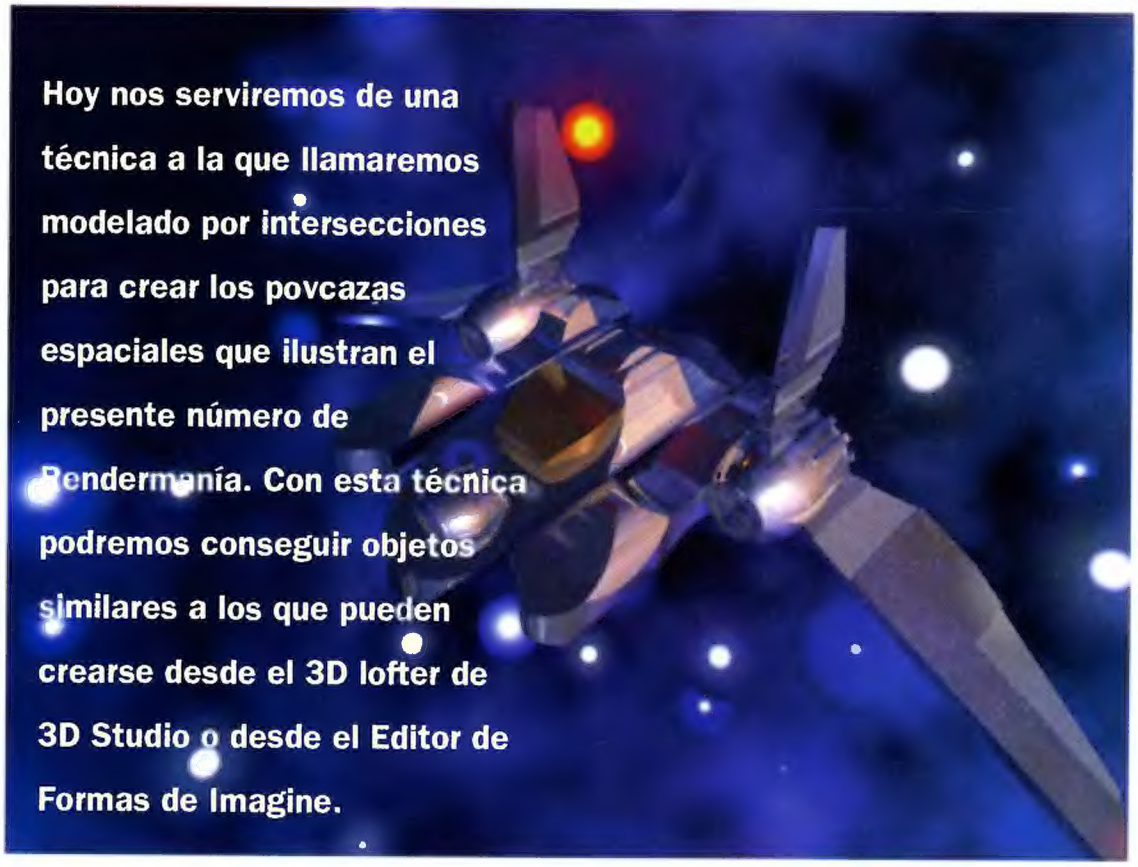
Magic Camera

LA PORTADA

Creamos ESTRELLAS y nebulosas con halos de emisión

pcmanía

Modelado por intersección



Hoy nos serviremos de una técnica a la que llamaremos modelado por intersecciones para crear los povcazas espaciales que ilustran el presente número de Rendermanía. Con esta técnica podremos conseguir objetos similares a los que pueden crearse desde el 3D lofter de 3D Studio o desde el Editor de Formas de Imagine.

El sistema que se emplea en los mencionados módulos de 3D Studio e Imagine consiste en utilizar dos o tres formas bidimensionales para crear una forma tridimensional, correspondiendo cada una de las formas 2d empleadas al perfil del objeto 3d, tal como éste puede verse al seguir paralelamente la línea de alguno de los ejes. En las escenas de Rendermania hemos aceptado siempre –arbitrariamente– que el "suelo" está en el plano X-Z. Por lo tanto, si el objeto va a ser construido descansando sobre dicho suelo, el eje Y sería el empleado para la llamada vista superior (o "top") del objeto a crear.

Supongamos que deseamos crear la forma básica de un coche (que reposará sobre nuestro suelo X-Z.) Para nuestros propósitos precisaremos de tres formas 2D para los tres perfiles del vehículo; uno con el perfil que presenta el modelo tal como lo vemos desde arriba (siguiendo el ejeY), otro con el perfil que exhibe el coche cuando lo vemos de

para cazas espaciales

frente (vista frontal) y otro con el perfil lateral del modelo (vista lateral). Aquí supondremos que el espectador está situado de modo que mira al objeto en construcción a lo largo del eje Z. Por tanto los ejes usados para las vistas será Y para la superior, Z para la frontal y X para la vista lateral.

Modelado usando perfiles

Hay muchos paquetes que disponen de opciones para crear objetos a partir de perfiles. Casi todos ellos funcionan a base de polígonos y quizá por esta razón sus modeladores presentan restricciones en cuanto a las formas 2d que pueden aceptarse para "alzar" los modelos. En algunos paquetes el dibujo para la vista top ha de ser siempre una forma convexa, en otros, los vértices de las vistas frontal y lateral deben cumplir ciertos requisitos, etc.

En cuanto al procedimiento que sigue el modelado por perfiles, es –en casi todos los programas poligonales– el siguiente. Usando los vértices de las vistas frontal y lateral, el algoritmo marca las alturas en las que deben crearse los vértices tridimensionales. Luego, para cada una de estas alturas, los vértices 2d del perfil top generan una sección tridimensional al adaptarse al volumen espacial resultante de la intersección con los vértices de los otros dos perfiles.

Esta poderosa técnica de modelado es la base para crear objetos cuyo diseño resultaría bastante difícil con otros métodos. Pero no todos los programas disponen de opciones para generar objetos con más de un perfil. Este es el caso de POV, de Polyray, y de otros programas de este tipo. Sin embargo, no hay que desesperar: existe un truco para hacer algo parecido al modelado de perfiles y tanto POV como Polyray pueden usarlo.

La extrusión es la solución

Los Povmaniacos más antiguos recordarán que antes de la versión 3.0, POV no disponía de ninguna sentencia que le permitiera generar objetos extruidos. Sin embargo, a pesar de esto, los usuarios de POV lográbamos crear objetos de apariencia extruida usando la sentencia heightfield, normalmente empleada para generar mallas de apariencia montañosa. Ahora, de la misma manera, podemos emplear un pequeño truco para conseguir objetos generados a partir de perfiles, aunque el lenguaje de POV no tiene una sentencia específica para ello.

La idea consiste en realizar una sencilla operación de intersección entre

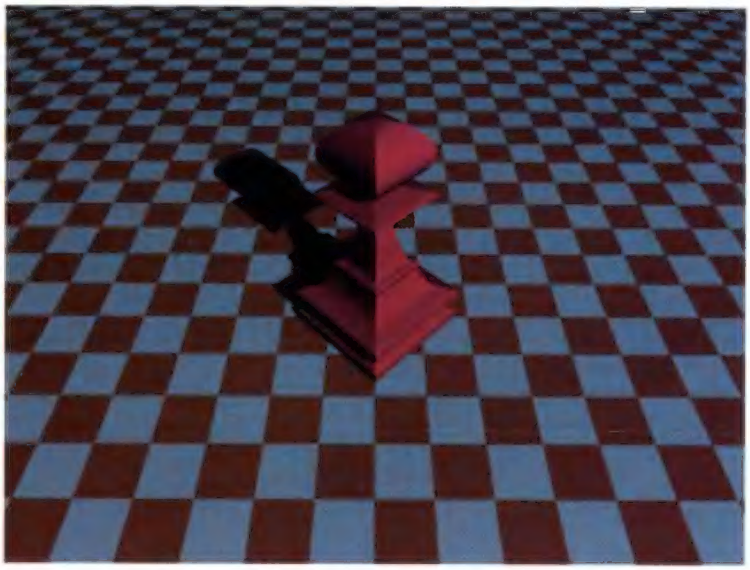
dos o más objetos extruidos. Por si aún hay alguien que no lo sepa, se llama objeto extruido o extrude a un objeto tridimensional generado mediante el procedimiento de desplazar una forma bidimensional a lo largo de un "path" o camino. Ejemplos sencillos de extrusión son el cubo que obtenemos al extrusionar (perdón, Real Academia de la lengua) un cuadrado o la letra tridimensional que conseguimos al dar profundidad a la forma 2d de dicha letra. En estos ejemplos, el path empleado es una línea recta perpendicular al plano donde está dibujada la forma 2d.

Este método llamado extrusión puede dar como resultado objetos muy simples o muy complejos, dependiendo de la forma y el path empleados. Si por ejemplo la forma 2d a extrusionar (con tu permiso otra vez Real Academia) es un círculo y el path una línea enrevesada, el resultado 3d podría ser una cuerda. La sentencia que implementa POV para realizar extrusiones es Prism y la sintaxis de la misma es:

```
prism {
  [ linear_sweep | conic_sweep ]
  [ linear_spline | quadratic_spline | cubic_spline ]
  ALTURA1,
  ALTURA2,
  numero_total_de_puntos,
  <punto_1>, <punto_2>, ..., <punto_n>
  [ open ]
  [ sturm ]
}
```

Con la sentencia Prism crearemos objetos 3d extrusionando una o más formas 2d a lo largo de un eje. Estas formas 2d iniciales se definen mediante listas de puntos incluidas dentro de la misma sentencia y deberán ser siempre cerradas. POV creará la forma 2d inicial conectando los puntos definidos con líneas o curvas y después extrusionará (elevará, desplazará) la forma resultante a lo largo del eje Y, con lo que el objeto 3d final quedará paralelo al eje X-Z. Más adelante estudiaremos la forma en que nos afecta esta orientación.

En la sintaxis de Prism, las palabras entrecorchetadas son opcionales. Las dos primeras se refieren al path y, por defecto, la usada por POV es linear_sweep, la cual da como path una línea recta perpendicular al plano X-Z. (El autor de estas líneas no tiene aún muy claro el funcionamiento de conic_sweep.) El siguiente parámetro a indicar controla el contorno de las formas 2d. Por defecto la opción empleada es linear_spline, lo cual da como resultado que los puntos utilizados para definir la forma 2d se conecten con líneas rectas. Esto significa que el objeto final será una forma 3d facetada y con contornos sin suavizar. Este inconveniente, no obstante, puede ser eludido empleando quadratic_spline o cubic_spline, que son dos tipos de curvas spline soportadas por POV para propósitos de suavizado. Como recordaréis, los splines son curvas generadas por fórmula cuya forma es determinada por uno o más puntos de control. Los splines se emplean bastante en los modeladores poligonales, en los cuales son usados para crear paths y dibujar formas 2d (para extrusionarlas luego). El uso de quadratic_spline o cubic_spline afectará además a la sintaxis anterior.



Los dos parámetros siguientes son las alturas Y donde comenzará y terminará la extrusión del objeto. Si por ejemplo la forma 2d es un cuadrado de 2*2 unidades y damos los valores 3 y 5 como alturas, el resultado 3d será un cubo de 2 unidades de lado que flotará a 3 unidades de altura sobre el suelo X-Z (situado en Y=0). Estos valores controlan pues, la longitud de la extrusión y la posición final del objeto 3d. El siguiente parámetro es el número total de puntos a cuyo valor, en caso de emplear splines, habremos de sumar los puntos de control usados para las curvas.

Después viene la lista de puntos de la forma 2d. Como se ha dicho anteriormente, si estamos empleando splines hay que indicar los puntos de control necesarios. Cuadratic_spline sólo necesita un punto –que colocaremos al principio de la lista– mientras que cubic_spline necesita dos: el primero lo pondremos al principio de la lista de puntos y el segundo al final de la misma. En cuanto a estos puntos de control, el autor de las presentes líneas reconoce que no sabe predecir la manera en que determinan la forma de la curva, debiendo poner los valores a ojo en cada prueba. (Normalmente lo mejor es emplear valores pequeños y cercanos a <0,0> para que las curvas no sean muy exageradas).

Además, la lista de puntos incluida en Prism puede describir varias formas 2d y en caso de que las siguientes estén inscritas dentro de la primera forma 2d, se producirán restas CSG en la misma. También hay que recordar que cada una de estas formas –llamadas subprismas en la documentación de POV– debe estar cerrada. Esto lo lograremos repitiendo al final el primer punto del subprisma. En cuanto a los puntos propiamente dichos, deben ser colocados siguiendo un orden determinado.

La siguiente palabra, open, es opcional y elimina las tapas superior e inferior del objeto extruido, dejándolo hueco. Naturalmente el uso de open impide efectuar posteriores operaciones CSG sobre el objeto, ya que este no tendrá un volu-

En las dos imágenes de esta doble página se aprecia como el cruce de la sección del peón produce este resultado con cubic_spline.

men interno con el que operar. Finalmente, si estamos empleando splines cubic y la visualización es incorrecta, podemos incluir la palabra sturm, que cambiará el algoritmo de cálculo para emplear el método de Sturmian, más lento pero más preciso.

Cómo elevar

Las tediosas explicaciones precedentes pueden haber sumido a los lectores más novatos en un estado de desesperación, pero realmente el uso de prism es sencillo. Lo primero que debemos hacer es crear la lista de coordenadas con la forma 2d. Para ello es imprescidible dibujar previamente dicha forma en un papel cuadriculado, lo que nos servirá para tomar nota de las coordenadas, las cuales anotaremos en la lista siguiendo un orden determinado. Una vez concluida la lista añadiremos los valores del número de puntos y las alturas y hecho todo esto estaremos en condiciones de visualizar la primera prueba. Para dicha prueba, lo mejor será colocar la cámara situándola en las coordenadas X-Z del centro de gravedad de la forma 2d y a una altura Y adecuada. (A esta posición de prueba para la cámara la llamaremos cámara top y la usaremos para comprobar la corrección del prism empleado.) Después enfocaremos la cámara apuntándola al centro del objeto extrusionado y listo. (Bueno, tambien es conveniente agregar la palabra "orthographic" al final de la declaración de la cámara para que no nos confunda la perspectiva.) Como siempre, lo mejor para saber utilizar algo es hacer muchas pruebas. Los lectores que desconozcan el tema pueden experimentar con los sencillos ficheros-ejemplo de POV o con el ejemplo del peón o de los modelos de cazas que estudiaremos unas líneas más abajo.

Modelado por intersección de perfiles

La noción de modelado por perfiles es muy simple una vez que se comprende lo que es la extrusión. Como sabemos, hay objetos que pueden construirse a partir de un único perfil. La extrusión de un cuadrado puede originar una caja, la de un círculo nos dará un cilindro, etc. (En todos estos ejemplos, el path debe ser una línea recta perpendicular a la forma). En POV todos estos objetos pueden crearse a partir de una sentencia prism sencilla pero ahora avancemos al siguiente nivel de complejidad.

Este lo forman los objetos que requieren dos perfiles para ser generados. Es el caso de las botellas, los peones de ajedrez, etc. Cada uno de estos objetos presenta el mismo perfil tanto frontal como lateralmente. De nuevo POV puede crear objetos de este tipo ya que cuando ambos perfiles son iguales, el objeto puede crearse como una superficie de revolución. ¿Pero y si el objeto tiene dos perfiles diferentes?

Este puede ser el caso de un coche (MUY sencillito) o del mismísimo ratón de nuestro ordenador. En el caso del ratón, por ejemplo, necesitaremos un vista superior y otra lateral (con la que obtendremos la curvatura con la que el ratón se adapta a la mano.) Ya hemos dicho antes que POV no dispone de ninguna sentencia que permita construir un objeto a partir de dos o más perfiles. ¿Qué hacer entonces? La respuesta es:

1) Crear un objeto extrusionado para cada perfil.

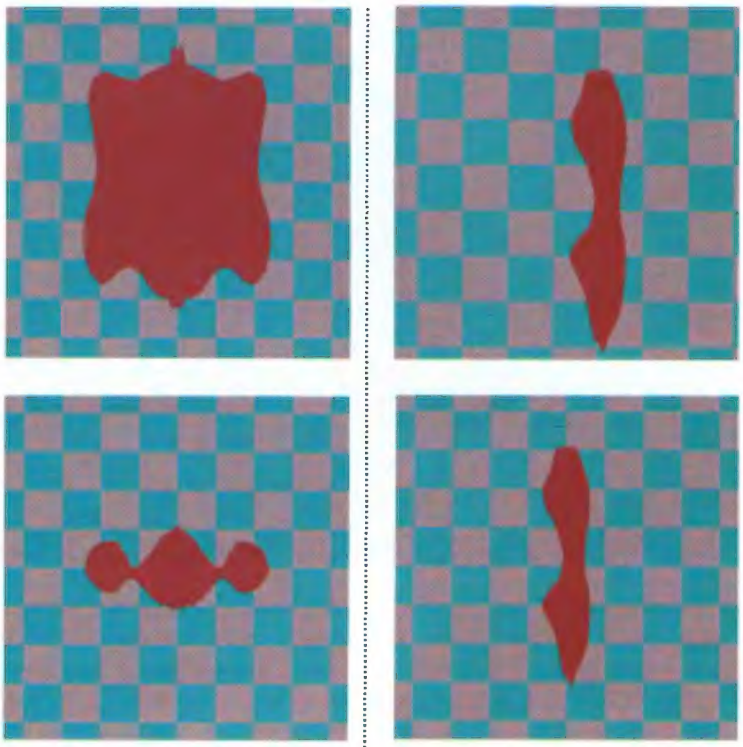
2) Rotar y situar adecuadamente los objetos extrusionados con vistas a la operación 3.

3) Crear un objeto CSG como resultado de la intersección de los objetos extrusionados.

Recordemos que en una operación CSG de intersección entre dos o más objetos, el resultado es siempre un objeto cuyo volumen espacial era el que

compartían todos los objetos implicados en la operación. Por ello también podríamos llamar a este truco "modelado por volúmenes". En fin, realmente esta técnica no produce exactamente los mismos resultados que pueden esperarse del Fit de 3D Studio o del Editor de Formas de Imagine (y que nos perdonen todos los usuarios de otros programas que también permiten modelado de perfiles: no es cuestión de presentar una lista completa). Sin embargo, por el momento, es lo más aproximado a ello que podemos lograr con POV.

Para intentar comprender las particularidades de lo que hemos llamado modelado por intersección, estudiemos un par de casos. Supongamos que queremos crear el modelo de un juguete: un cochecito de madera sumamente sencillo que sólo precisa emplear dos perfiles, el superior y el lateral (ruedas y volante aparte, claro). Para ello extrusionaremos los perfiles superior y lateral del cochecito y, después de orientarlos y situarlos adecuadamente, crearemos el cochecito con una operación de intersección entre ambos objetos extrude. Esto no parece revestir dificultades. Pero veamos otro caso más complejo. Supongamos ahora que queremos crear un peón de ajedrez desde POV usando esta técnica. (Sería una tontería ya que disponemos de Sor, pero es un buen ejemplo.) Dibujaremos el perfil del objeto y escribiremos la lista de puntos correspondiente en una sentencia Prism. Como este perfil corresponde tanto a la vista frontal como a la lateral del peón, lo mejor será utilizar un #declare para poder referenciar dos veces el prism declarado. Acto seguido crearemos el objeto extrude. Si observamos el objeto con la cámara de



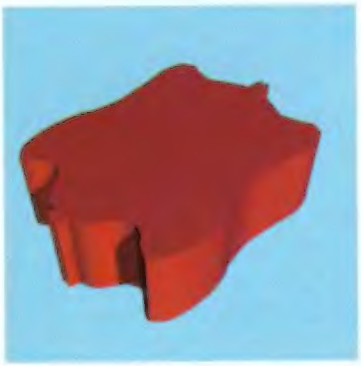
Vistas superior, lateral (del centro de la nave), frontal y lateral (de los lados) de la nave.

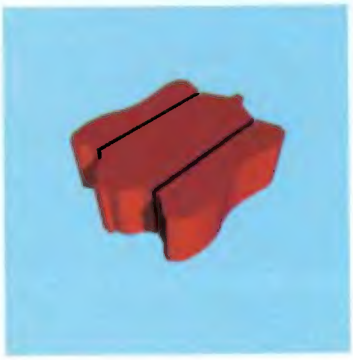
prueba top descrita anteriormente apreciaremos el perfil del peón, si sus puntos se han escrito correctamente. Si se aprecian fallos graves en la forma, la causa estará casi con toda seguridad en algún error de la lista de puntos o bien de un fallo en la predicción de las curvas splines, si las hay.

El siguiente paso será colocar el objeto extruido en la orientación necesaria, o sea, en este caso, con la base del peón descansando en el suelo X-Z. Para ello efectuaremos una rotación de 90 grados en el eje X para el objeto. De esta manera obtendremos el perfil frontal para la intersección. Si hemos dado las coordenadas de los puntos con astucia, de modo que los puntos correspondientes a la base del perfil estén en Z=0, ni siquiera tendremos que desplazar la forma extrusionada para colocarla sobre el suelo.

Ahora vamos a por el perfil restante. Crearemos un nuevo objeto con el mismo perfil pero esta vez efectuaremos dos rotaciones. La primera, en el eje X, servirá como antes para dejar el nuevo perfil con la base en el suelo y la siguiente, de 90 grados en el eje Y, nos servirá para obtener la vista lateral del objeto. Se entiende que para que esto último funcione, la forma 2d inicial del peón habrá de definirse centrada en el eje Z. De no ser así habremos de efectuar maniobras superfluas de traslación, a fin de superponer adecuadamente ambos objetos.

El último paso es muy sencillo. Usando ambos objetos crearemos el peón





realizando una intersección entre ambas formas. Podéis ver el resultado de todo esto en las fotos (y realizar vuestras propias pruebas con peon.pov). Como veis el objeto resultante, aunque interesante, no es exactamente un peón. ¿Por qué? (El autor de estas líneas reconoce que, al principio, pensaba que el fruto de estas operaciones sería un peón.) La respuesta está en el hecho de que estamos ante una operación booleana (AND) entre los volúmenes espaciales de dos objetos-perfil idénticos. En ella cada objeto-perfil se extiende a lo largo de un path cuya longitud ha de ser en este caso el ancho del peón. Al principio es difícil comprender lo sucedido pero basta con examinar atentamente las fotos para entender el principio. La técnica requiere cierta capacidad de imaginación espacial ya que habremos -mentalmente- de descomponer el objeto deseado en sus perfiles antes de dar un solo paso. Por otro lado, como acabamos de ver, no podremos conseguir cualquier forma. Con otros modelos, en cambio, la intersección de volúmenes funcionará muy bien –¿recordáis las almenas de los castillos del número 1?– Veamos otro caso: la construcción de un caza espacial.

¿Cazas espaciales?

Podría parecer que debe haber pocas cosas que requieran menos reglas que el diseño de naves espaciales futuristas. A fin de cuentas el artista puede dibujarlas tal y como le de la gana, sin que nadie pueda chistarle, pues no existen proporciones ni modelos de comparación fuera de las ilustraciones de SF.

(Nota del Redactor Jefe: José Manuel Muñóz desapareció misteriosamente mientras estaba preparando este artículo. Días más tarde volvió a aparecer por la redacción y manifesto que. en el momento de hallarse sumido en la elaboración de las líneas precedentes, fue raptado por entidades extraterrestres y llevado a un sitio indeterminado fuera de nuestro planeta. Al parecer estas entidades habían captado las furiosas emanaciones mentales de este articulista, el cual se hallaba desesperado porque no conseguía imaginar una sola nave espacial decente. Ansiosas de ayudarle, las entidades le mostraron sus naves espaciales por dentro y por fuera sólo para ver, atónitas, como José Manuel Muñóz rechazaba de plano sus diseños aduciendo que no podía presentar en Rendermanía modelos tan ridículos y simplones.

Las mencionadas entidades extraplanetarias se enfurecieron entonces muchísimo ante este rechazo a miles de años de adelantos tecnológicos, y procedieron a llevar a cabo diversos experimentos con el cerebro de J.M.M.P, tras lo cual le liberaron dejándole en un estado lamentable. Por esta razón no podemos garantizar la validez del resto del presente artículo.)

Pruebas de iluminación.

Por ejemplo, el artista puede imaginar que su nave emplea los medios conocidos de propulsión y dibujar cohetes. O bien puede decidir que la nave utiliza la presión de la luz y optar por un velero solar. Incluso puede aducir que su creación emplea la energía de la improbabilidad –igual que la "Corazón de Oro" de "Guía del autoestopista galáctico"– y dibujar una nave rodeada por monos escribiendo las obras de Shakespeare. La libertad es total.

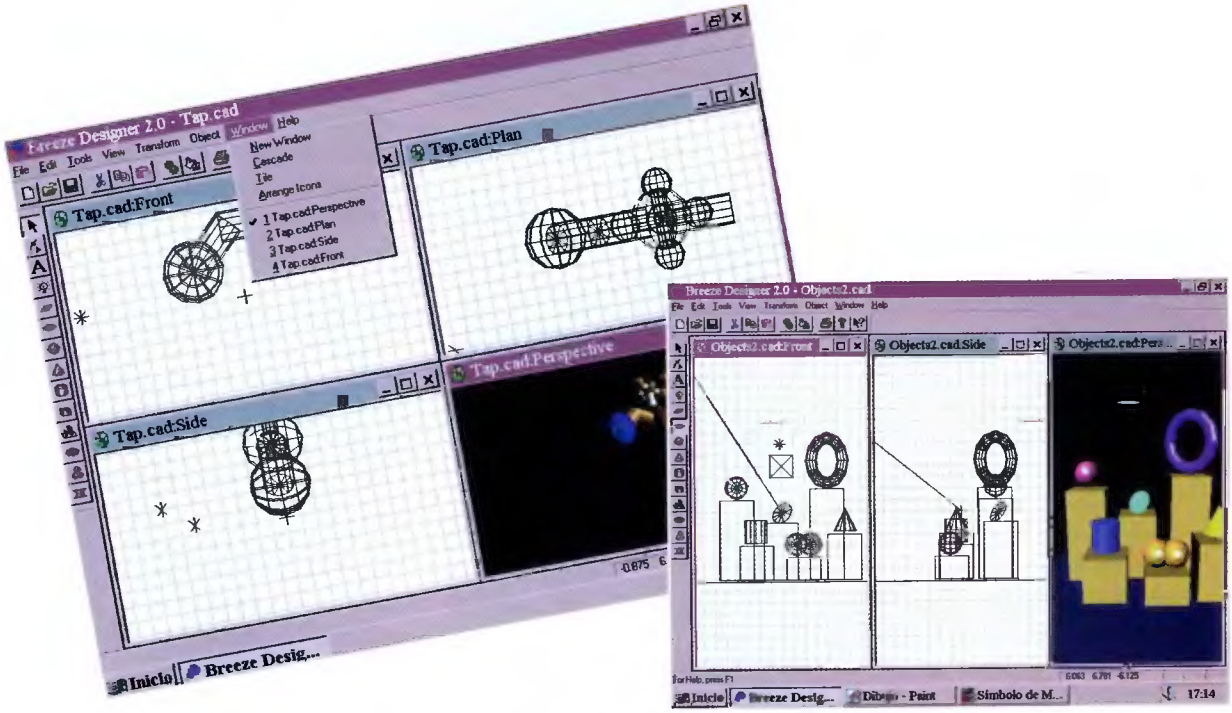
Sin embargo a la hora de la verdad la cosa no es tan sencilla. El caso es que el artista desea que su obra parezca una nave espacial futurista y... ¿cómo puede lograrlo si no hay un modelo en el que basarse? Si se basa demasiado en los cohetes actuales, el resultado no parecerá demasiado avanzado tecnológicamente hablando y si se usa demasiado la imaginación el resultado puede acabar pareciendo cualquier cosa menos una nave espacial. El problema puede agudizarse si queremos crear un tipo espacial de nave, por ejemplo un caza. Un caza espacial no necesita alas pero si no se las ponemos es probable que el modelo no nos parezca un caza. Tampoco debería necesitar una cabina de cristal como las de los cazas terrestres, ya que en el espacio profundo los objetos que no emiten luz no son visibles (excepto por el hecho de que ocultan las estrellas, por supuesto). Pero, de nuevo, el hecho de poner una cabina nos hará creer que el caza tiene unas dimensiones reducidas, tal como el espectador espera.

Hroki1

Este es el nombre de nuestro caza. La estructura principal del mismo está fuertemente basada en la de un coche deportivo que sirvió como modelo; –como se ha dicho antes, cualquier cosa puede servir de inspiración para crear un caza espacial: el autor vio en cierta ocasión una nave sumamente parecida a un WC con alitas y reactores varios–. Si el lector examina las fotos con las formas a extrusionar y la secuencia de construcción comprenderá enseguida como se ha llegado al resultado final.

En el caza hay dos cuerpos principales: el central, que requirió tres formas extrude (para los planos superior, frontal y lateral) y el lateral, en el que se usó un perfil distinto. Aquí el mayor problema fue encajar estas dos piezas lo cual (¡Ay!) no se resolvió demasiado bien ya que no se consiguió una transición suave. Luego se empleó otra intersección de extrudes para la cabina y se añadieron los propulsores (hechos con Sor) y las alas (más extrusión.) Esperemos que, con más experiencia, los próximos modelos creados con esta técnica resulten mejor.

Moray 2.5 vs Breeze 2.0



En el número 46 de Pcmanía asistimos a una pequeña comparativa entre dos de los mejores modeladores existentes para raytracers: Moray y Breeze. El primero, funcionando bajo el arcaico MS Dos, es ya un viejo conocido de muchos de nuestros lectores. Breeze, en cambio, es un programa casi desconocido en estos lares a pesar de su espectacularidad y de su cómodo interface. Hoy presentaremos dos nuevas versiones de ambos programas para intentar responder a las mismas preguntas de aquel entonces: ¿Podrá Breeze desbancar a Moray? ¿Merece la pena emplear un modelador?

Los rendermaniacos más antiguos conocen la reticencia del autor de estas líneas a emplear modeladores. Las razones son múltiples aunque difíciles de explicar. La mayoría de los trazadores de rayos construyen sus universos virtuales mediante objetos generados con fórmulas, no con polígonos. Este es el caso de POV, Polyray, Vivid, Magic Camera y muchos raytracers más. Estos objetos generados a base de fórmulas implican un procesamiento visual más lento que en el caso de los objetos generados por polígonos y por esta razón los programas mencionados no incorporan un modelador interno para diseñar las escenas. En lugar de esto, como sabemos, estos programas requieren del empleo de editores de texto para dicha tarea. Editores con los que escribimos ficheros fuente compuestos por las sentencias permitidas por el lenguaje escénico de cada programa.

Problemas de los modeladores

El caso es que el empleo de un buen modelador suele ser más rápido e intuitivo que el de cualquier lenguaje escénico. Además hay muchos usuarios que no quieren "programar" escenas. Para ellos hay cuatro posibles opciones:

A) Crear la escena con un modelador comercial potente y usar una utilidad de conversión para "traducir" el fichero del modelador al formato escénico del raytracer y poder generar la escena con él.

B) Usar alguno de los modeladores –de coste bajo o nulo– desarrollados para estos raytracers.

C) ¡Aprender el lenguaje escénico!

D) Prescindir de POV, Polyray, etc.

Como mucha gente no está dispuesta a pasar por C, no disponen de A y no quieren llegar a D, la única opción restante es B. Pero los modeladores de la opción B no tienen la misma orientación ni potencia que los de A. Los modeladores de bajo costo como Moray o Breeze (aquí los llamaremos modeladores-B) han sido creados expresamente para trabajar con POV y otros raytracers. Trabajan mediante un sistema de ventanas y visualizan polígonos como los modeladores comerciales pero ahí se acaba el parecido. Programas como Moray y Breeze utilizan las mismas primitivas disponibles en los lenguajes escénicos pero empleando objetos poligonales en las ventanas de trabajo. Al acabar la sesión, sin embargo, los objetos poligonales deben ser "traducidos" a sentencias del lenguaje escénico para que el raytracer empleado pueda generar la escena creada desde el modelador. Esto implica los siguientes problemas:

1) Las fórmulas necesarias para resolver las operaciones CSG no son idénticas para los objetos poligonales y los generados por fórmula y son muy complejas. Hasta ahora ello ha significado que los objetos CSG no se visualizan adecuadamente en las ventanas de trabajo de modeladores como Moray y Breeze.

2) Los modeladores-B van siempre a remolque de las versiones nuevas de los trazadores de rayos.

3) Existen muchas primitivas que no se visualizan correctamente en la mayoría de los modeladores-B. Este es el caso de los objetos blob y los heightfields.

4) El trabajo con los modeladores-B no es demasiado ágil ya que el motor de render está fuera. Por tanto o bien hay que llamar al raytracer desde el modelador o bien hay que salir de éste y ejecutar la escena exportada con el raytracer para, después, retomar al modelador.

5) El uso de uno de estos modeladores no implica que podamos olvidarnos del lenguaje escénico ya que muchas de las posibilidades del raytracer deben ser controladas –aunque se ordenen desde el modelador– con conocimiento de los efectos de dichas sentencias. Por ejemplo la niebla, la atmósfera, etc.

6) Las sentencias de programación –if, while... de POV–, algunas funciones, etc., deben usarse fuera del modelador.

Sin embargo, a pesar de todos estos inconvenientes, los modeladores seguirán siendo usados por aquellos que quieran crear modelos concretos para sus escenas o que, sencillamente, odien la "programación escénica".

La nueva beta de Breeze

En el número 46 de Pcmanía examinamos la versión 2.0 Beta de Breeze y la que hoy comentamos sigue siendo una beta -aunque distinta- de la misma versión. Breeze, programado por Neville Richards tiene una serie de características entre las que destaca el soporte de OpenGL y la librería de render 3DR de Intel. En concreto el uso de OpenGL nos permitirá visualizar modelos sombreados y texturas aplicadas en objetos desde las ventanas de Breeze sin necesidad de invocar al raytracer. Esta posibilidad es, junto con la sencillez de manejo del

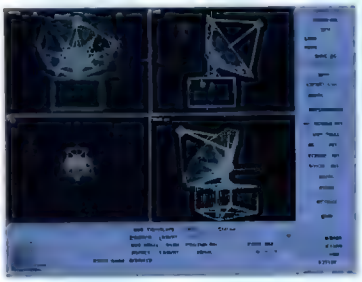
programa y la ayuda, quiza lo más atractivo del programa pero, por ahora, sigue sin resolverse el tema de la visualización de las diferencias e intersecciones CSG.

En la nueva versión Neville declara haber arreglado algunos fallos y haber implementado soporte para exportación de escenas en VRML y RenderMan. Aparte de esto, Breeze ya soportaba importación de ficheros 3DS y DXF, agrupaciones de objetos, llamada a POV sin abandonar el programa, superficies de revolución y extrusión, etc. El programa -que no incluimos aquí por tratarse aún de una beta- tendrá en sus futuras versiones un sistema de render por Red con el cual será posible utilizar un grupo de ordenadores para ejecutar un render o una animación y extensiones para el macrolenguaje (¿?). En resumen: pocos cambios entre las dos betas.

La versión 2.5 de Moray

Ya que Moray ha sufrido más cambios que Breeze vamos a dedicarle algo más de espacio. La instalación resulta muy sencilla. Basta con descomprimir el fichero zip en un directorio específicamente creado para Moray y ejecutar el setup. Al descomprimir se crearán una serie de subdirectorios dentro del nuestro donde se guardarán las utilidades, los ejemplos, etc. Es muy importante no olvidar poner los paths adecuados para llamar al render y referenciar a los ficheros de inclusión del render. Si no lo hacemos así u olvidamos pulsar F9 para grabar el setup podemos volvernos locos intentando averiguar por qué fallan las llamadas al render.

En esta nueva versión de Moray, sus autores han incluido soporte para todas o casi todas las nuevas características de POV 3.0. Claro que esto quiere decir en muchos casos que se han incluido menús para controlar opciones de POV, no que Moray realice cálculos propios. Tal es el caso de la atmósfera, la niebla, la radiosidad, etc. Por otro lado Moray sigue siendo un programa de DOS –que funciona en modo protegido bajo Vesa–, pero no por mucho tiempo. En los mismos ficheros de la actual versión ya se anuncia que la próxima será un programa de 32 bits para Windows 95 y NT. ¿Se incluirá visualización flat en dicha versión? Esperemos que sí. Por ahora éstas son algunas de las mejoras de la última versión:



-Soporte para objetos blob y Superellipsoid.

-Soporte para las nuevas posibilidades de la cámara.

-Soporte para los nuevos objetos de revolución y extrusión.

-Mejorado el soporte a Polyray

-Posibilidad de llamar directamente al Render, en máquinas de mucha memoria. (¿?)

-Mejoras en la edición de objetos.

-Mejoras en el redibujado de objetos.

-Código para visualizar objetos CSG. (Según afirman, aún no muy fiable).

-Posibilidad de ver bitmaps como fondo.

Quizá lo peor de Moray sea que su interface, aunque rápido y eficaz, es, en algunas pantallas, bastante lioso y poco intuitivo: uno llega a asustarse de la cantidad de opciones que tienen algunas ventanas. En cuanto al botón de render, sigue siendo más práctico abandonar el programa y ejecutar el render desde fuera ya que la cantidad de memoria necesaria es ridículamente prohibitiva; en una escena sencilla ejecutada en un ordenador con 64 megas de Ram, la llamada directa a POV implicó un inacabable ronroneo del disco duro durante media hora mientras la misma escena ejecutada exteriormente requirió menos de un minuto. También hay que decir que el fichero moray.ini de configuración admite demasiados comandos. Sólo su explicación ya requeriría varias páginas (aunque por otra parte también es posible que no necesitemos tocarlos). Todo esto no ha cambiado desde la versión anterior.

En resumen

No es probable que los usuarios experimentados que antes no usaran modelador cambien su filosofía por culpa de estas nuevas versiones. Por otro lado no hay que confiar demasiado en cambios revolucionarios (en lo que se refiere a la visualización) para la próxima versión de Breeze. Pero quizá si podría haberlos en la de Moray, ya que el paso de Dos a Windows implica muchos cambios en la gestión de memoria (optimización de llamadas directas) y en la visualización (¿OpenGL? ¿Direct3D?). Por ahora, en nuestra opinión, Moray sigue siendo la mejor opción.

Caza estelar Hroki

modelo LG-700

Un solitario caza estelar Hroki aguarda, oculto en el centro de la nebulosa donde se esconde su base, la aparición de algún desprevenido mercante procedente de un cercano agujero de gusano. Naturalmente también podría aparecer una flota imperial dispuesta a limpiar el sector de piratas. Pero esto ya ha ocurrido antes y nuestro caza Hroki, aunque ya algo anticuado, confía en la potencia de su motor trucado –pero imprevisible– que le convierte en uno de los montones de basura más rápidos de la galaxia, con permiso del Halcón Milenario, por supuesto.

Al menos esto es lo que diría el autor de esta nave, y nadie puede demostrar lo contrario. ¿Quién puede afirmar que, en algún lejano sistema estelar, no existe una nave como ésta? ¿Y por qué dicha nave no podría disponer del más moderno condensador de fluzo, de un exprimidor de protones ultracorto, de un recargador de ñazos y de un colector de...? En fin, en un Space Opera todo está permitido. Los motores hacen ruido en el vacío del espacio y las naves son perfectamente visibles en el espacio profundo. Incluso en obras más rigurosas, el tema espacial se permite una fantasía (en los diseños) que no solemos encontrar en otros campos y por ello los ilustradores tienen las manos mucho más libres para trabajar.

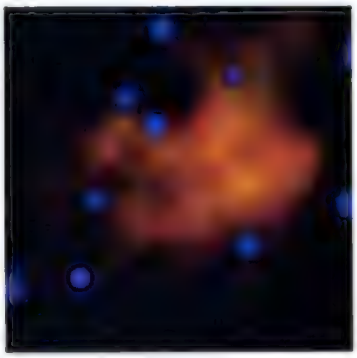
Portadas posibles

Desde que el autor de estas líneas tuvo ocasión de ver aquel magnífico fondo estelar de Felix Higueras Jorna, la idea de preparar alguna portada con fondos estelares hechos con POV quedó decidida. Los pormenores de la construcción de la nave empleada en la portada de hoy ya han sido desvelados en las páginas anteriores pero la idea inicial era diseñar un gigantesco crucero espacial. Para ello se podrían haber usa-

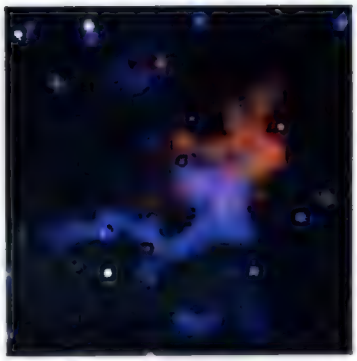
do las sentencias de programación de POV para crear un modelo compuesto por cientos de objetos menores dispuestos en jerarquías, tal como se hizo en el número dedicado a la construcción de castillos medievales, pero la magnitud del proyecto pronto resultó excesiva para el tiempo disponible. Para empezar no se deseaba construir escenas que requiriesen un consumo de memoria excesivo, como en el caso de los castillos, y además existían otras dificultades: es difícil hacer comprender al espectador el tamaño de un objeto si no hay otros objetos con los que compararlo. Lo ideal sería incluir escuadrones de cazas a diferentes distancias y poner algunos cerca del crucero para demostrar el enorme tamaño de éste. Todo esto ya implicaba la creación de otro modelo cuyos requisitos de desarrollo eran diferentes a los del crucero. Así pues se optó por crear un caza estelar y construir la escena en torno a él.

Aquí también había jugosas posibilidades entre las que elegir. Una idea –ya bastante exprimida pero siempre atractiva– fue crear un campo de asteroides donde aparecieran varias naves luchando. Esto requería básicamente dos cosas: la primera, por supuesto, un conjunto de objetos-asteroide y la segunda una manera de colocar espacialmente a los mismos. Los asteroides pueden conseguirse de varias maneras distintas. Quizá la mejor sea emplear alguna utilidad de fractalización de objetos a fin de conseguir la forma irregular y caprichosa de estos objetos. Estas utilidades funcionan del siguiente modo: toman la geometría de un objeto inicial simple como un cubo o una esfera y aumentan su nivel de detalle paso a paso. Con cada paso desplazan también sus vértices

El fondo estelar de nuestra portada.



Una prueba con halos glowing.



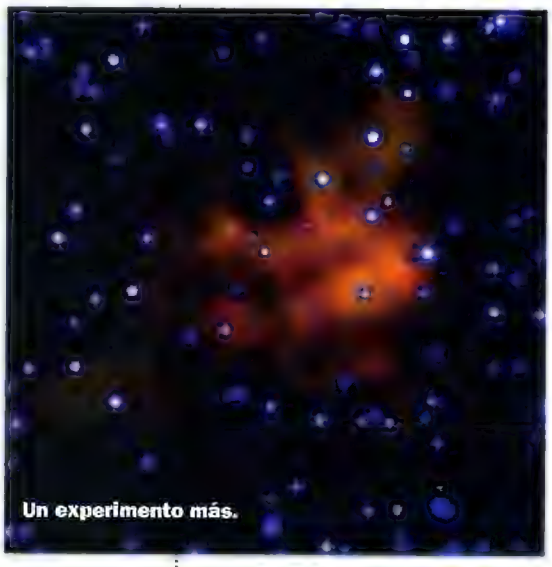
Aquí puede apreciarse lo que sucede cuando chocan los colores de estrellas y nebulosas.

al azar con lo que el objeto va tomando una forma irregular. Un valor pequeño de "ruido" da a los objetos una apariencia gastada, sin cambiar demasiado su forma, mientras que un valor mayor en dicho factor produce objetos muy irregulares. El primer inconveniente que tiene esto para nosotros es que estas utilidades trabajan con objetos poligonales. Por supuesto, es posible, si conocemos el formato ascii que acepta la utilidad (si es que éste es el caso), que podamos definir nosotros mismos el objeto pero esto puede ser tedioso. Afortunadamente existen utilidades freeware como Frgen de Steve Anger -que Pcmanía ya publicó- que pueden servirnos.

Frgen tiene varios ficheritos ascii de ejemplo donde se describen cubos, esferas, etc. Frgen puede crear, partiendo de estos objetos simples iniciales, objetos irregulares. De un cubo podemos obtener un cubo de hielo o un asteroide y de una esfera un planetoide. Frgen puede, asimismo, grabar los objetos obtenidos en el formato de POV o en el de Polyray. Una vez en POV, después de importar el objeto, podremos escalarlo y cambiar sus dimensiones. El único problema es que Frgen no calcula las normales del objeto creado, lo cual quiere decir que la apariencia del mismo será facetada. Claro que también podemos emplear otra utilidad para calcular estas normales.

De todos modos la idea del campo de asteroides fue también rechazada debido a consideraciones de memoria. Si queremos conseguir un campo de asteroides realista, hay que usar docenas –o mejor cientos– de objetos poligonales de todos los tamaños. La colocación de estos objetos es sencilla, gracias a las nuevas sentencias de programación de POV. Podemos disponer

Un ejemplo perfecto de todos los fallos que pueden cometerse.


Un experimento más.

que los objetos se distribuyan de manera aleatoria (con rand) por un volumen espacial determinado. Aún en caso de que uno o más objetos se superpongan (lo cual no tiene por qué suceder si la densidad de objetos no es excesiva), ello no tiene que quedar mal, ya que estaríamos trabajando con asteroides, los cuales pueden tener cualquier forma. Este método de colocación 3d de objetos sólo supone un pequeño paso más con respecto al que ya usamos en los números 0 y 1 de Rendermanía, para colocar casas y soldados y es el que hemos empleado para la distribución estelar de la portada.

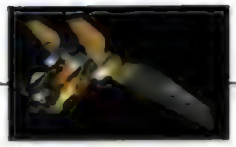
La textura estelar

Una vez descartados los asteroides, por ahora, la idea final fue la que podéis ver en la portada del presente número. La escena emplea tres elementos como fondo: una textura estelar, un campo de estrellas-halo y un par de nebulosas también creadas con halos. El fondo de estrellas más pequeñas (los puntos) no es más que una textura de granito que incluye un mapa de colores. POV incluye un fichero llamado Stars.inc, donde hay definidas varias texturas de este tipo. La "densidad estelar" de estas texturas oscila desde el poco denso Starfield1 hasta el máximo Starfield6. Nosotros hemos empleado Starfield3 colocando la textura sobre un plano Z al que la cámara enfoca perpendicularmente.

El cúmulo estelar

La textura anterior sólo proporcionó un fondo de puntos que apenas servía como fondo para nuestro caza. Era preciso añadir estrellas más cercanas o de mayor tamaño. Esto fue posible gracias a la sentencia Halo cuyo análisis ya incluimos en el número anterior de Rendermania. Como recordaréis los halos no son más que texturas que simulan una distribución de partículas dentro de un objeto que sirve como contenedor. Estas partículas pueden absorber o emitir luz y de hecho nuestro cúmulo será visible aunque desactivemos las luces de la escena. Las estrellas fueron creadas con halos de emisión, ya que este tipo pareció el más adecuado para lograr la apariencia deseada. Al menos así parece deducirse de las pruebas realizadas con los otros tipos de halos.

Siguiendo el ejemplo dado por Félix, aquí empleamos dos halos para cada estrella; en la primera se genera el tenue halo luminoso que se extiende fuera de la estrella y en la segunda se genera la estrella-halo propiamente dicha. La idea inicial era colocar cientos de estrellas en el fondo, de hecho el contador "numsoles" está inicializado a 200. Pero pronto se comprobó que un número excesivo de estrellas de este tipo no daba un buen resultado, debido al modo en que éstas interactuaban con las nebulosas del fondo (aquí es donde podéis comprobar la diferencia entre el uso de uno u otro tipo de halo). Estudiando la escena de Felix comprobaremos que el autor colocó las estrellas manualmente y con cuidado para que éstas no se superpusieran a las nebulosas, con lo cual se evita este problema. Esto sin embargo puede ser tedioso



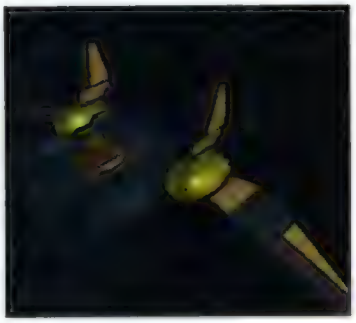
(sobre todo si uno está cambiando la cámara con frecuencia), y por ello se optó por la distribución al azar.

Porta53.pov emplea mucho la sentencia rand para dar valores aleatorios a las variables que se emplean para posicionar a las estrellas y para decidir su color. Por ello puede suceder que muchas escenas no sean útiles al presentar fallos visuales, ya que es imposible predecir si el paisaje estelar va a estar en armonía con el fondo de la nebulosa. En estos casos bastará con cambiar el valor S de la semilla del generador.

Los fallos mencionados consisten en que el halo de emisión de las estrellas "choca" con el fondo de las nebulosas. Siendo más exactos: lo que sucede es que la nebulosa puede hacer visible a la esfera contenedora más externa de la estrella. Es un fallo que no siempre resulta evidente pero que puede ocurrir, sobre todo si empleamos nebulosas cuyo color sea demasiado diferente al de la estrellas.

Examinemos las líneas de porta53.pov que generan el cúmulo estelar. En primer lugar se crea una semilla para el generador de números aleatorios usado para decidir el color de las estrellas y situarlas en el espacio. Esta semilla es muy útil ya que cambiándola podremos generar un paisaje estelar distinto y asimismo, si no nos gusta el nuevo paisaje, podremos recuperar el antiguo simplemente volviendo a poner el valor de semilla que lo genera. La siguiente variable es el contador que emplea el bucle #while para crear las estrellas. La definición de éstas se realiza dentro del bucle y su colocación se decide con las operaciones...

```
#declare px=(int(rand(S)*1000)-500)
#declare py=(int(rand(S)*1000)-500)
#declare pz=(int(rand(S)*1000)-500)
```



Diferentes pruebas de texturas aplicadas a nuestro caza.

Estas operaciones nos darán una posición aleatoria <px, py, pz>, comprendida entre la zona espacial definida por <-500, -500, -500> y <500, 500, 500>. En cuanto al color de la estrella se decidirá según el valor aleatorio depositado en la variable col. Según sea éste, las variables r, g, b, r2, g2 y b2 tomarán valores de color que se usarán en los dos halos de la estrella.

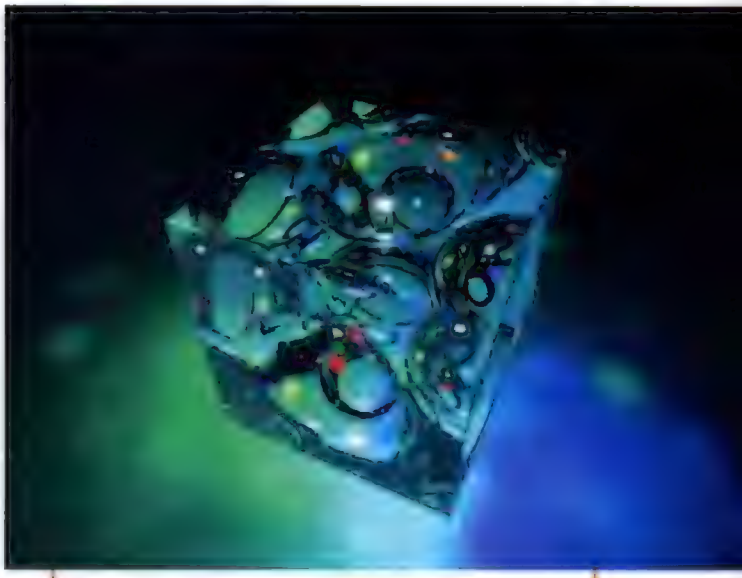
Nebulosas

Las dos nebulosas empleadas en la escena están colocadas dentro de grandes esferas de cientos de unidades de diámetro centradas en el eje de coordenadas. Como tanto el caza como la cámara están muy cerca de esta posición, el resultado es que la "foto" está tomada desde dentro de la nebulosa. Las estrellas pueden quedar dentro o fuera de este radio. Un detalle que tiene su importancia ya que parece que sólo las estrellas que caen dentro del volumen espacial de las nebulosas pueden presentar fallos graves de visibilidad (las otras, aparecerán más difuminadas, lo cual parece natural en el paisaje interno de una nebulosa.)

Para crear escenarios estelares diferentes podemos cambiar los valores turbulence de los halos, la escala de los objetos contenedores y la orientación de los mismos. El resultado de cada uno de estos cambios es imprevisible a priori pero afortunadamente el tiempo de generación no es excesivo, lo cual nos deja bastante tiempo para hacer pruebas. Otra idea interesante sería colocar las estrellas de otro modo, con un valor de distribución menor en un eje, a fin de conseguir un plano galáctico (mayor densidad estelar). También podríamos haber colocado una o dos estrellas realmente grandes, un planetoide, etc.

Visita a un Museo virtual

Hoy descargamos nuestro buffer de misivas. Los lectores envían escenas y animaciones de Objetos extraños, naves, ritos satánicos, plataformas marítimas, chicas manga en 3d, aviones y muchas creaciones virtuales más, creadas con diversos programas. Sólo para Rendermaniacos.

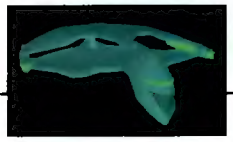


Tomas Garralaga Camarero también se ha subido al tren de POV y nos envía unas primeras imágenes bastante resultonas entre las que destaca la sencilla pero hermosa escena boxal2. ¡Gracias por tus ánimos a Rendermanía y sigue así tú también!

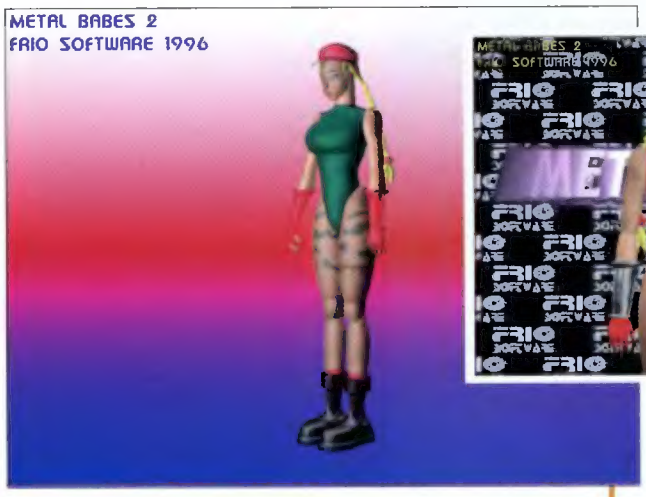
Julián Pérez Serrano, un viejo conocido en estas páginas, ataca de nuevo, esta vez con un F15e Eagle modelado precisamente con la misma versión 2.5 de Moray que hoy incluimos en el CD. Gracias por tus felicitaciones a mis castillitos virtuales y tomo nota de tu apoyo a la idea del concurso de infografía (aunque dicha idea aún esté algo verde, me temo) y de tu sugerencia de potenciar la búsqueda de puntos 3d interesantes en la red. Siguiendo con tu carta, siento decirte que tienes la negra: primero la reciente catástrofe en tu disco duro y luego el desastre en los flexibles que has enviado. ¡Todos tus zips daban errores! Afortunadamente dejaste fuera un par de imágenes que servirán para incluirte en el foro de hoy. Todo lo demás -tu carta al foro, los fichero mdl, etc.- no es recuperable (Sorry). En cuanto a lo que preguntas del efecto Lens Flare, POV no lo tiene que yo sepa pero puedes usar el de Polyray y mezclar la imagen con la de POV (ya que también te gusta trastear con programas 2D). Muy resultón tu cacharro.

No dejes de leer Julián, porque **Abraham Limpo Martínez** -alias Volcano-, un maquetador aficionado a los aviones quiere indicarte algunas cosillas sobre tu Fokker. Reproducimos algunas: 1) El Fokker empleaba cartucheras con cadenas de balas y las ametralladoras comenzaban con una forma cuadrada seguida de un cilindro de refrigeración. No llevaban punto de mira puesto que éste estaba situado entre ambas ametralladoras. 2) La caja del motor no se pintaba. 3) Las palas de la hélice eran algo más pequeñas. Esto no debe tomarse como una crítica -afirma Abraham- sino sólo como unas aclaraciones de otro gran aficionado a los aviones.

Hablando de otra cosa, **Pablo Costas** nos envía un e-mail comentando que desde hace algún tiempo intenta mantener una página Web sobre POV en castellano. ¡Buena idea! En ella hay faqs y tutoriales traducidos y se espera que siga creciendo con la aportación de todos. La página está en: http://www.arrakis.es/~pcostas/povray



Nota importante: Podéis remitirnos vuestros trabajos o consultas, bien por carta a la dirección que figura en la segunda página de Pcmanía, o vía e-mail a rendermania.pcmania@hobbypress.es



Os recomendamos que leáis en El Foro la carta de **Juan Miguel González.** En ella este miembro de Frío software expone que busca gente para crear juegos, gráficos, etc.

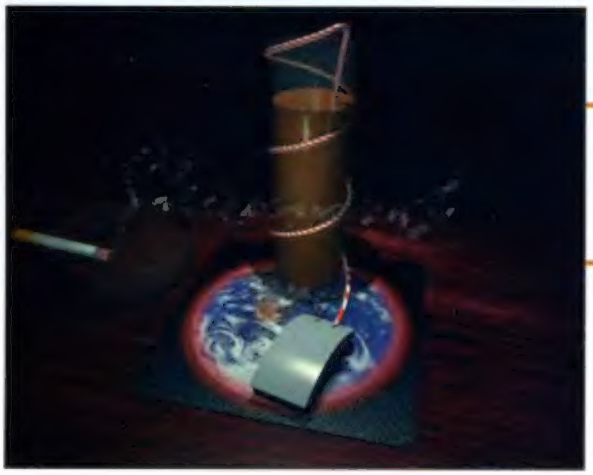
Juan Miguel nos envía hoy unos modelos creados con 3D Studio basados en un personaje de Street Fighter: Cammy. Juan admite que sus chicas aún no son tan buenas como las mangababes de Tomwoof pero "todo llegará". Mientras tanto podéis encontrar sus imágenes y las mallas correspondientes en el directorio que lleva su apellido. No os perdáis las caricaturas.

Ricardo Villanueva Yoldi nos ha enviado de nuevo las escenas que en el mes anterior resultaron machacadas. Esta vez las imágenes llegaron felizmente. En ellas Ricardo saca un buen partido de Povafx para renderizar los modelos creados con Moray. Hay naves espaciales y castillos. ¡No os las perdáis! En cuanto a tu pregunta, si puedes enviar los ficheros por carta en vez de por e-mail, la respuesta es que sí.

Antonio José García Mora es un nuevo povmaniaco que nos envía una carta en la que plantea dos problemas principales. El primero radica en lo incompleto de los manuales de POV, en los cuales no es extraño ver líneas "en elaboración" en apartados cruciales. Aquí desconocemos la fecha en que el Povteam se dignará completar el manual. De hecho la versión de Dos y la de Windows tienen diferencias importantes. En cuanto a tus problemas para generar las escenas de castillos del número 1 de Rendermanía, los mensajes de error que describes son todos de falta de memoria. Como ya se indicó entonces, muchos de los ejemplos requerían una memoria considerable. De hecho cada lancero requería unos 7 megas. Cuando POV se queda sin Ram, intenta coger memoria del disco duro y el problema puede radicar en que no haya suficiente. Hay un apartado en la pantalla de render donde se indica la memoria total que necesita POV para generar la imagen. Comprueba si este valor no es superior al de tu memoria libre. (Yo he generado escenas que requerían más de 80 Megabytes). Prueba a montar castillos con menos elementos –en el futuro intentaré que las escenas no sean tan costosas–. Con respecto a tu pregunta, si leiste el número anterior de este suplemento, ya sabrás que POV no tiene luces flare. ¿Cómo simular este efecto? Como he dicho más arriba, una posibilidad sería generar el flare con Polyray (que sí tiene este efecto) y mezclar las escenas con alguna utilidad 2d.

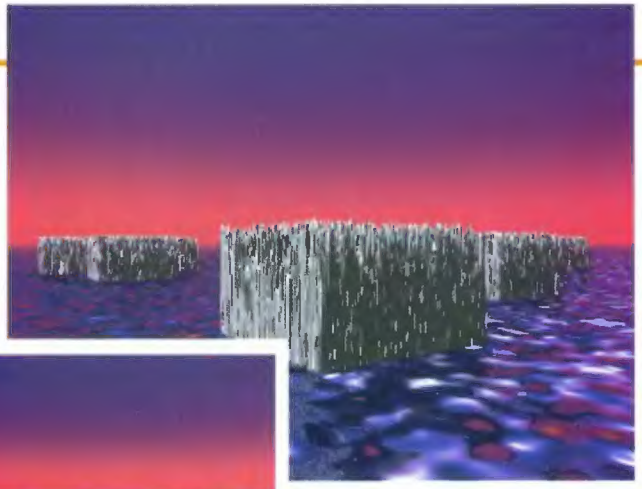
El Foro del lector

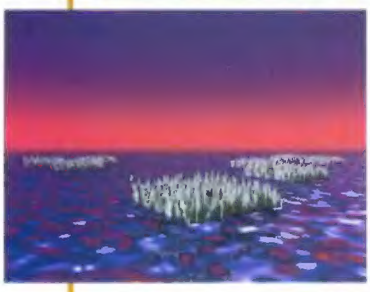
Alberto Riera Sánchez -alias Che- nos envía unas cuantas imágenes creadas con 3D Studio de las que solicita una pequeña crítica. Bien, ahí va. Me gustan el color, la atmósfera y la fantasía que has conseguido en Alquimia, Interior y Marte (sobre todo en esta última escena). Parece ser que en una crítica anterior te dije que tus mallas eran muy sencillas. Quizá me expresé mal. El que un modelo sea sencillo no tiene por qué ser necesariamente malo. Todo depende del contexto: un modelo antropomórfico no puede ser sencillo -a menos que sea una caricatura- pero una nave espacial sí puede serlo y quedar bien. A lo que probablemente me refería era a que necesitabas coger mayor soltura en el modelado, aspecto este en el que pareces haber mejorado bastante. Lo único que no me ha convencido ha sido la torre de la escena "naves". Tomo nota de tu apoyo a Rendermanía en general y a la idea del concurso en particular, y transmito tus felicitaciones a Felix Higueras por su sonda espacial y a Juan Miguel González por sus chicas metálicas.



Luis Martínez Huguet pide opinión sobre su obra Vaso2. Sin embargo no me aclaras con que está hecha -lo cual es vital para valorarla- y tampoco envías los ficheros de generación. En fin, lo menos que puede decirse es que la imagen tiene detalles interesantes y es divertida.

Hoy va de naves espaciales. Nuestro amigo **Nacho** nos envía otra que se desplaza sobre un bonito escenario marítimo con puesta de sol incluida. Buena escena salvo por la textura de la nave que, para ser sinceros, me ha parecido espantosa (pero sobre gustos no hay nada escrito). Por otro lado es cierto lo que dices acerca de las fotos del primer Rendermanía: salieron negras y la culpa (¡Ay!) fue mía por no haber dado unos colores menos apastelados o no haberlas tratado antes de enviarlas a la Imprenta. Espero que no vuelva a suceder (¡sniff!).





Carlos Cortés Salinas es otro povmaniaco recientemente enganchado a este mundillo. Nos envía su dos primeras imágenes –icebergs generados con heightfields– de las que solicita una crítica. Pues bien amigo Carlos, la única crítica importante está en la tga que has empleado para crear el heightfield y en la escala y disposición de éste. Como ya sabes, los pixels de la tga determinan la altura de la malla en el objeto heightfield resultante. Si se emplean medios algorítmicos para crear la tga, los pixels de los bordes no tienen porque diferir de los centrales, con lo que la altura media será la misma en todos los puntos de la tga. Por eso, como la altura de la malla cae a 0 en los bordes de la misma, el resultado es que, si enfocas todo el objeto como lo has hecho, verás un corte cuadrado regular bastante impropio en un objeto fractal. (Leéte la explicación que viene en "La Portada" del número1). Lo primero debería ser crear una tga con picos algo más suaves que la que has usado y luego escalarla adecuadamente para que no se vieran los picos externos cortados. (Las dos islitas de la portada del número 1 son un heightfield). En cuanto a tus preguntas, que yo sepa no hay planes para publicar un libro de la versión 3.0 de POV. El módulo spline editor de Imagine por el que preguntas (y que yo no he usado) debería servir para crear formas bidimensionales con curvas spline. Estas formas pueden ser empleadas luego como paths o como formas iniciales de una extrusión para crear un objeto 3d.

REFUGIO

Julián Hierro García ha dejado -como el mismo dice- de ser un lector pasivo para convertirse en un povmaniaco activo. Para demostrarlo Julián nos envía sus dos primeras imágenes. ¿Crítica? No están mal para ser las primeras. El siguiente paso será practicar, practicar y practicar el modelado. Publicamos tu carta en el foro. Hasta pronto.

Jose María Hernández Borja nos envía un disco Zip con una película de ciencia ficción completa montada a base de ficheros fli. Entre las secuencias hay combates entre naves espaciales, mechs, efectos, robots, etc. Hay buenos efectos y detalles y nada malo que decir sobre ella excepto una cosa: 58 megas es demasiado espacio para nuestro CD. Esto nos ha impedido, desgraciadamente, publicar la película completa, debiendo limitarnos a seleccionar unos cuantos ficheros fli. Sorry Jose María.

Todo los ficheros de inclusión y los ejemplos podéis encontrarlos en el directorio PCMANIA\RENDER53 del CD-Rom

Magic Camera

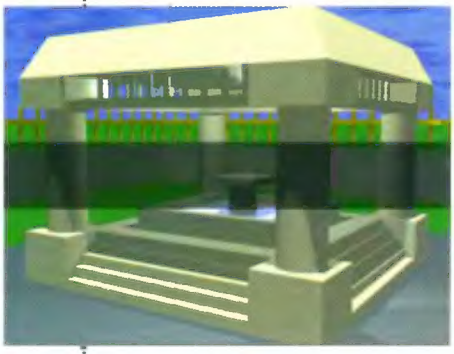
Hoy presentamos un nuevo trazador de rayos. Se trata de la versión shareware de Magic Camera de Dan Wesnor, un programa procedente del mundo de Amiga, en el que las escenas no se crean mediante un modelador, sino a través de un lenguaje propio.

La versión que hoy publicamos es un programa Windows de 16 bits. La versión registrada es un programa de 32 bits que funciona -según el autor- un 35% más rápido que la versión sin registrar. La instalación del «Magic Camera» es sencilla usando la opción -d de Pkunzip que creará los directorios con los scripts de ejemplo y otros ficheros. El uso del programa tampoco es difícil: Pulsando sobre él aparecerá una ventana desde donde podremos especificar el fichero a renderizar y las opciones deseadas. Desgraciadamente, esta es una versión limitada. La renderización de imágenes de más de 100.000 pixels tendrá como resultado el añadido de una fea banda horizontal en la imagen.

El programa incorpora un fichero hlp de ayuda que resultará más bien parco para alguien acostumbrado a los detallados manuales de POV y Polyray. Aparte de este manual, el paquete incluye tres ficheros que, supuestamente, nos servirán de ayuda: scan.l. que contiene una lista con las palabras soportadas por el lenguaje y expresiones validas, Parse.y, que contiene la gramática del lenguaje y defaults.c que indica los valores por defecto para lo que Wesnor llama sub-elementos. ¿No hubiera resultado más sencillo agruparlo todo en un único fichero más completo? ¿Por qué no se indica lo que hace cada sentencia junto a la gramática? (Si hay un infierno, Wesnor debería ser condenado a estudiar su propio doctorial.)

Las pocas y escuetas indicaciones sobre el lenguaje de «Magic Camera» las encontraremos en el apartado "Writing Scripts" del manual. Según parece el programa usa las características a que nos tienen acostumbrados otros programas; texturas procedurales, primitivas, funciones, variables, etc. Para ser justos, incluso un programa como POV carecía hasta hace poco de algunas de las funciones de «Magic Camera».

Sin embargo «Magic Camera» también tiene ciertos detalles de bastante e incluso mucho interés. Existe una sentencia, por ejemplo, con la que podemos "destruir" los objetos que ya no van a emplearse en una animación. Con esto los objetos inútiles se borrarán de la memoria y podremos aprovecharla para otros.



Otro punto interesante es la posibilidad de definir objetos-hijo. Estos objetos se definen como "hijos" de otros, lo cual resulta muy útil para hacer animaciones o componer posiciones de objetos complejo. ¡Qué lo disfrutéis!

José Manuel Muñoz